

ICSS I2C FIRMWARE DESIGN GUIDE

I2C PRU Firmware

Applies to Product Release: 01.01.00.00
Publication Date: September 11, 2018

Document License

This work is licensed under the Creative Commons Attribution-NoDerivs 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nd/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

Contributors to this document

Copyright (C) 2018 Texas Instruments Incorporated - <http://www.ti.com/>



Texas Instruments, Incorporated
20450 Century Boulevard
Germantown, MD 20874 USA

This document is intended for users who are interested in getting more detailed understanding of the firmware design. It discusses ICSS based I2C firmware implementation details along with any features added on top of the basic I2C firmware i.e. SMBUS supports. It mentions the memory maps, structures and software design flow of the firmware.

Note: Those who just want to use ICSS I2C firmware may not need to go through this document

Revision History			
Version	Date	Description of changes	Author(s)
1.0	04-Jan-18	Initial version	Suraj Das
1.1	09-Sep-18	Added details on AM437x ICSS0 feature enhancement	Frank Livingston

TABLE OF CONTENTS

LIST OF FIGURES	6
LIST OF TABLES	7
1 INTRODUCTION	8
2 FEATURE SET	9
3 DESIGN DESCRIPTION	10
3.1 DESIGN LAYOUT	11
3.1.1 <i>Register Memory Map</i>	11
3.1.2 <i>Register Description</i>	12
3.1.2.1 I2C_COMMAND	12
3.1.2.2 I2C_BUF.....	13
3.1.2.3 I2C_CNT.....	14
3.1.2.4 I2C_CON	14
3.1.2.5 I2C_SA	14
3.1.2.6 I2C_PRU_PIN	14
3.1.2.7 I2C_PRU_CMD_CODE.....	15
3.1.2.8 I2C_PRU_INST_ID.....	15
3.1.2.9 I2C_PRU_TX_DATA.....	15
3.1.2.10 I2C_PRU_RX_DATA	15
3.2 DESIGN CHALLENGE	16
3.2.1 <i>Pinmuxing using PRU</i>	16
3.2.2 <i>Pinmuxing using EDMA</i>	16
3.2.3 <i>Using IEP DIGIO pins</i>	17
3.3 PRU RESOURCE USAGE	18
3.3.1 <i>PRU Data RAM</i>	18
3.3.1.1 Configuration Memory Region	18
3.3.1.2 Instance Memory Region	18
3.3.2 <i>Local Register</i>	18
3.3.3 <i>Scratchpad</i>	19
3.4 DESIGN THEORY	20
3.4.1 <i>Initialization Task</i>	20
3.4.2 <i>Scheduler Task</i>	22
3.4.3 <i>Communication with host</i>	24
3.4.3.1 Interrupt support for Host.....	25
3.4.4 <i>I2C protocol states</i>	27
3.4.4.1 Reset.....	28
3.4.4.2 READY.....	29
3.4.4.3 DATA Transfer	30
3.4.4.4 Read SCL.....	32
3.4.4.5 Reset Slave.....	33
3.4.5 <i>Concurrent execution</i>	34
3.5 FIRMWARE SOURCE CODE.....	35
3.5.1 <i>Firmware Macros Description</i>	35
3.5.2 <i>Firmware Sources Description</i>	36
4 RTOS DRIVER SUPPORT	37
4.1 EXTERNAL APIS	37
4.2 INTERNAL FILES.....	38
5 TEST PLANS	40
5.1 EVM SUPPORT.....	40
5.1.1 <i>Icev2AM335x</i>	40
5.1.2 <i>idkAM437x</i>	40

5.1.3	<i>idkAM572x</i>	40
5.1.4	<i>idkAM574x</i>	40
5.1.5	<i>idkAM571x</i>	40
5.1.6	<i>iceK2G</i>	40
5.2	EXTERNAL I2C BOARD.....	41
5.2.1	<i>I2C EEPROM Board</i>	41
5.2.2	<i>I2C and SMBus IO Expander Evaluation Module</i>	42
5.3	TEST SETUP.....	42
5.3.1	<i>Icev2AM335x</i>	42
5.3.2	<i>idkAM437x</i>	42
5.3.3	<i>idkAM572x</i>	43
5.3.4	<i>idkAM574x</i>	43
5.4	UNIT TEST.....	44
6	FIRMWARE FEATURE ENHANCEMENT	44
6.1	MODIFICATIONS TO I2C FIRMWARE FOR AM437X ICSS0.....	44
6.2	FEATURES AND LIMITATIONS OF AM437X ICSS0 I2C FIRMWARE.....	45
6.3	I2C FIRMWARE RESOURCE REQUIREMENTS.....	45
6.3.1	<i>Memory Requirements</i>	45
6.3.2	<i>PRU Cycle Count Requirements</i>	45
6.3.2.1	<i>I2C_FW, AM437X ICSS1</i>	46
6.3.2.2	<i>I2C_FW_AM437X_ICSS0, AM437X ICSS0</i>	47
6.4	TEST PLAN.....	48
6.4.1	<i>EVM Support</i>	48
6.4.1.1	<i>idkAM437x</i>	48
6.4.2	<i>Test Setup</i>	49
6.4.2.1	<i>idkAM437x</i>	49
6.4.3	<i>Unit Test</i>	50

LIST OF FIGURES

Figure 1. I2C FW Design Layers	10
Figure 2. pinmux switching timing diagram.....	16
Figure 3. Electric connection for SDA line	17
Figure 4. Init Task.....	20
Figure 5. scheduler state function	22
Figure 6. HOST sends command to PRU	24
Figure 7. PRU reads command from HOST.....	24
Figure 8. PRU responds back to HOST	25
Figure 9. Host reads response from PRU.....	25
Figure 10. Global State Diagram	27
Figure 11. RESET State.....	28
Figure 12. Ready state transtion.....	29
Figure 13. General Data Transfer Flow	31
Figure 14. Read SCL Line	32
Figure 15. RESET slave.....	33
Figure 16. Scheduling graph for firmware.....	34
Figure 17. I2C EEPROM Test Board	41
Figure 18. SMBus Expander module	42

LIST OF TABLES

Table 1. Feature Comparison Hard Vs Soft IP	9
Table 2. I2C firmware Register Memory Map	11
Table 3. Register Description for I2C firmware	12
Table 4. FW Command Values	13
Table 5. FW Command Response	13
Table 6. Buffer Size encoding	14
Table 7. Configuration Memory Map	18
Table 8. CPU Register Usage	19
Table 9. Scratchpad Memory Usage	19
Table 10. Macros List	36
Table 11. Firmware Source List.....	36
Table 12. List of Application APIs	37
Table 13. Function Description of APIs	37
Table 14. Firmware Internal APIs	39
Table 15. iceAM335x I2C Instances	40
Table 16. idkAM437x I2C Instances	40
Table 17. idkAM572x I2C Instances	40
Table 18. idkAM574x I2C Instances	40
Table 19. Test board pin details	41
Table 20. icev2AM335x Test Setup	42
Table 21. idkAM437x Test Setup.....	43
Table 22. idkAM572x Test Setup.....	43
Table 23. idkAM574x Test Setup.....	43
Table 24. I2C Firmware Memory Requirements	45
Table 25. I2C_FW Standard Mode Max. Cycle Counts (C_{max}) on AM437x ICSS1	46
Table 26. I2C_FW Full Mode Max. Cycle Counts (C_{max}) on AM437x ICSS1	47
Table 27. I2C_FW HS Mode Max. Cycle Counts (C_{max}) on AM437x ICSS1.	47
Table 28. I2C_FW_AM437X_ICSS0 Standard Mode Max. Cycle Counts (C_{max}) on AM437x ICSS1	47
Table 29. I2C_FW_AM437X_ICSS0 Full Mode Max. Cycle Counts (C_{max}) on AM437x ICSS1	48
Table 30. I2C_FW_AM437X_ICSS0 HS Mode Max. Cycle Counts (C_{max}) on AM437x ICSS1	Error! Bookmark not defined.
Table 31. idkAM437x I2C Instances	49
Table 32. idkAM437x Test Setup.....	50

1 Introduction

The I2C (Inter-IC) bus is a bi-directional two-wire serial bus that provides a communication link between integrated circuits (ICs). Most of TI's SoCs have their own dedicated hardware I2C IP. In case of need of more I2C instance then supported via hardware, firmware based I2C soft IP can be used.

Firmware is designed to run on PRU cores. PRU cores have their own GPI/GPO pins which can be toggled at specific time interval in order to implement I2C protocol. PRU programming is done using assembly instructions. PRU cores also have their EIP timer to meet the timing requirements. Host cores in SOC will configure PRUs for receiving and sending data on the I2C bus. Every ICSS has 2 PRU. Both PRUs will be able to run the I2C firmware independently. Also within a PRU, all instances will work independently then each other.

This Firmware design is not compatible with any of Ethernet based ICSS firmware design. It means one cannot load I2C firmware in one PRU for example PRU0 and Dual_emac Firmware in other PRU for example PRU1.

2 Feature Set

The following are the list of features which will be supported for I2C firmware. It also compares the features available on firmware with that available on hardware IP. Firmware features mentioned below are supported on both PRU0 and PRU1. For example, both PRU0 and PRU1 supported 2 instances of I2C which makes 4 instances in total.

I2C Supported Features	Hardware IP	Firmware (PRU0 & PRU1)
No. of hardware instance	SoC dependent	4 (Standard mode) 1 (Fast mode) 1 (HS mode at 1 MHz)
SMBus support	NO	YES
Addressing modes	7/10-bit	7/10-bit
Master mode	YES	YES
Slave mode	YES	NO
Combined Master-Slave mode/transaction	YES	NO
I2C data transfer rate (Standard / Fast / HS mode: up to 100 kbps / 400 kbps / 3.4 Mbps)	100 kbps / 400 kbps / 3.4 Mbps	100 kHz / 400 KHz / 1 MHz I2C clock frequency*
Bit format transfer	8 bit	8 bit
DMA support (one read DMA event and one write DMA event that the DMA can use)	YES	NO
Interrupts that the CPU can use	YES	1
Peripheral enable/disable capability	YES	YES
Start/Restart/Stop	YES	YES
Built-in configurable FIFOs (8, 16, 32, 64 bytes) for buffered read/ write	8/16/32/64	8/16/32/64/128/256
Programmable clock generation	programmable	NO
8-bit-wide data access	YES	YES
Slave reset feature	NO	YES
Internal loopback feature	NO	YES
Implement Auto Idle mechanism (SoC Specific feature)	YES	NO
Implement Idle Request/Idle Acknowledge handshake mechanism (SOC Specific feature)	YES	NO
Support for asynchronous wakeup mechanism	YES	NO

Table 1. Feature Comparison Hard Vs Soft IP

*Note: Maximum supported I2C clock frequency, maximum bit rate will be reduced by I2C protocol overhead.

3 Design Description

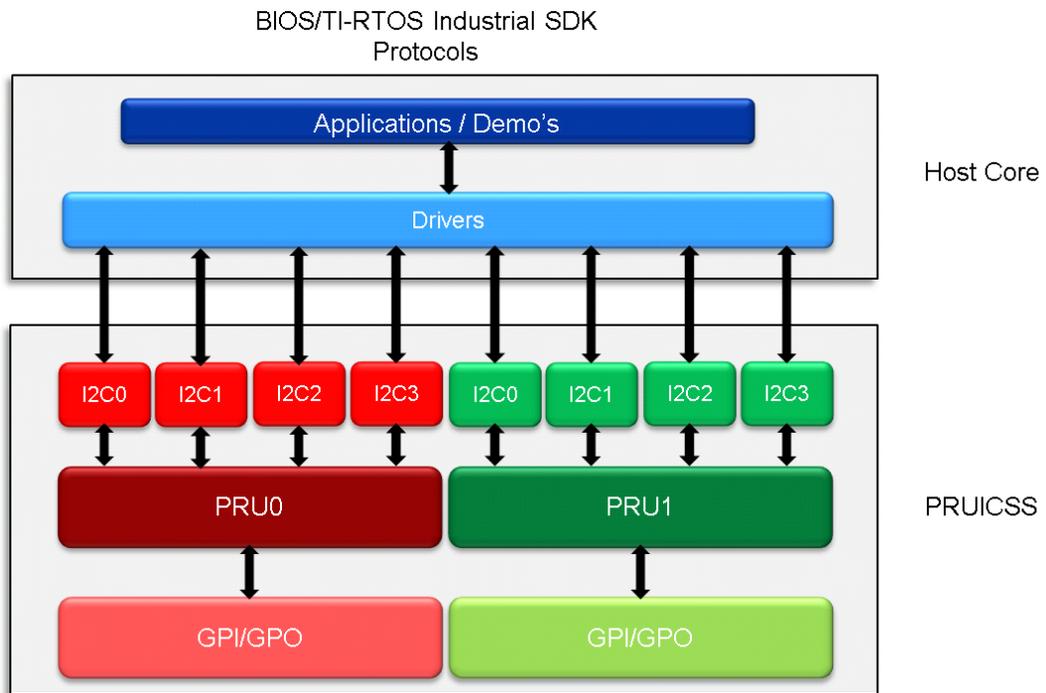


Figure 1. I2C FW Design Layers

3.1 Design Layout

3.1.1 Register Memory Map

The location of register memory map of each instance of I2C firmware will be in its respective PRU's Data memory. PRU0 will have it in DATA RAM0 and PRU1 will have it DATA RAM1.

SOC	Device	Module Instance	Module Base Address	Size (Bytes)
AM57xx & K2Gx*	ICSS0 PRU0	Configuration Memory	PRU0_DATA_RAM + 0x00000400	256
		I2C0	PRU0_DATA_RAM + 0x00000500	768
		I2C1	PRU0_DATA_RAM + 0x00000800	768
		I2C2	PRU0_DATA_RAM + 0x00000B00	768
		I2C3	PRU0_DATA_RAM + 0x00000E00	768
	ICSS0 PRU1	Configuration Memory	PRU1_DATA_RAM + 0x00000400	256
		I2C0	PRU1_DATA_RAM + 0x00000500	768
		I2C1	PRU1_DATA_RAM + 0x00000800	768
		I2C2	PRU1_DATA_RAM + 0x00000B00	768
		I2C3	PRU1_DATA_RAM + 0x00000E00	768
AM437x, AM335x, AM57xx & K2Gx	ICSS1 PRU0	Configuration Memory	PRU0_DATA_RAM + 0x00000400	256
		I2C0	PRU0_DATA_RAM + 0x00000500	768
		I2C1	PRU0_DATA_RAM + 0x00000800	768
		I2C2	PRU0_DATA_RAM + 0x00000B00	768
		I2C3	PRU0_DATA_RAM + 0x00000E00	768
	ICSS1 PRU1	Configuration Memory	PRU1_DATA_RAM + 0x00000400	256
		I2C0	PRU1_DATA_RAM + 0x00000500	768
		I2C1	PRU1_DATA_RAM + 0x00000800	768
		I2C2	PRU1_DATA_RAM + 0x00000B00	768
		I2C3	PRU1_DATA_RAM + 0x00000E00	768

Table 2. I2C firmware Register Memory Map

*Note: I2C firmware is not supported on ICSS0 for AM3.

3.1.2 Register Description

The following are the detailed description of the register values.

Register name	Offset	TYPE	Bits	Description
I2C_COMMAND	0x08	RW	31:16	Command Word
			15:0	Command Response
I2C_BUF	0x94	RW	15:8	Size of memory buffer in RX mode
			7:0	Size of memory buffer in TX mode
I2C_CNT	0x98	RW	15:0	Data Count
I2C_CON	0xA4	RW	15	I2C module enable
			13:12	Operation mode selection
			10	Master/slave mode
			8	Expand Slave address
			5	SMBUS Burst mode
			4	End SMBUS with ACK
			1	Stop condition (master mode only)
			0	Start condition (master mode only)
I2C_SA	0xAC	RW	9:0	Slave address
I2C_PRU_PIN	0xD8	RW	23:16	PRU GPO pin number for EDIO SDA
			15:8	PRU GPI pin number for SDA
			7:0	PRU GPO pin number for SCL
I2C_PRU_CMD_CODE	0xE0	RW	7:0	Command Code SMBUS mode
I2C_PRU_INST_ID	0xE4	RW	8:0	ICSS I2C instance id.
I2C_PRU_TX_DATA	0x100	W	256 Bytes	TX Data
I2C_PRU_RX_DATA	0x200	R	256 Bytes	RX Data

Table 3. Register Description for I2C firmware

3.1.2.1 I2C_COMMAND

Command Word: The value tells I2C firmware the next action to take. The following tables indicate the list of command used by I2C firmware.

Command	Value	Description
Reset cmd	0x10	This command sends the firmware into reset state. It drops all earlier configurations. Once firmware is in reset, setup_cmd needs to be passed again for configuring the firmware.
Setup cmd	0x11	This command configures the firmware based on the value it reads from the registers. Once this command is passed, firmware will start reading through all the mmap registers. All the mmap registers should updated first and then this command should be passed.
Rx cmd	0x12	This command starts to receive data on the line. It reads the data count and slave address values.
Tx cmd	0x13	This command starts to send data on the line. It reads the data count and slave address values.
Quick cmd	0x14	This is an SMBUS quick command.
Send byte cmd	0x15	This is an SMBUS command for sending a byte.

Recieve byte cmd	0x16	This is an SMBUS command for recieving a byte.
Write byte cmd	0x17	This is an SMBUS command for writing a byte.
Read byte cmd	0x18	This is an SMBUS command for reading a byte.
Write word cmd	0x19	This is an SMBUS command for writing a word (2 bytes).
Read word cmd	0x1A	This is an SMBUS command for reading a word (2 bytes).
Block write cmd	0x1B	This is an SMBUS command for writing a N bytes of data.
Block read cmd	0x1C	This is an SMBUS command for reading a N bytes of data.
Read scl	0x1D	This command is used for reading the SCL line value.
Reset slave	0x1E	This command will reset the slave device if it is hunged. It will follow standard I2C protocol for resetting the slave, if slave have kept the SDA line low.

Table 4. FW Command Values

Command Response: The value tells the response of I2C firmware for the last command passed. The following tables indicate the list of responses used by I2C firmware.

Response	Value	Description
Command success	0x0500	This response indicates that PRU was able to perform the command succesfully.
Reset command failed	0x0501	This response indicates that PRU was not able to bring out I2C firmware out of reset successfully.
Setup command failed	0x0502	This response indicates that setup_cmd was not successful.
Tx command failed	0x0503	This response indicates that Tx was not successful.
Rx command failed	0x0504	This response indicates that Rx was not successful.
Scl value high	0x0505	This response indicates that SCL line is high for 10 clock cycle when read for error condition.
Scl value low	0x0506	This response indicates that SCL line is low for 10 clock cycle when read for error condition.
Reset slave done	0x0507	This response indicates that Slave reset was done with 9 dummy clock pulse.
Address acknowldege failed	0x0508	This response indicates that No ACK was received after slave address was transmitted.
Data acknowldege failed	0x0509	This response indicates that No ACK was received after data was transmitted.
Master slave mode failed	0x050A	This response indicates that I2C mode is either incorrect or not supported.
Addressing mode failed	0x050B	This response indicates that I2C addressing mode is either incorrect or not supported.
Invalid command	0x050C	This response indicates a unknown command as been passed.
Invalid data count	0x050D	This response indicates the number of bytes to be sent/receive is more than buffer size.
Time out error	0x050E	This response indicates time out has happened.

Table 5. FW Command Response

3.1.2.2 I2C_BUF

This indicates the size of the memory buffer for RX and TX mode. Both TX and RX bit field 7 bits wide. It supports value of 8, 16, 32, 64, 128 or 256 bytes memory region. The following table shows the encoding of each buffer size.

Buffer Size	Encoding Value
8 Bytes	1
16 Bytes	2
32 Bytes	4
64 Bytes	8
128 Bytes	16
256 Bytes	32

Table 6. Buffer Size encoding

3.1.2.3 I2C_CNT

This indicates the amount of data in bytes to be sent/read.

3.1.2.4 I2C_CON

I2C module enable: indicates if the particular instance is enabled or not.

1-> module is enabled.

0-> module is disabled.

Master/slave mode: indicates which mode is i2c currently set to.

1-> Master mode.

0-> Slave mode (Not supported).

Expand Slave address: indicates slave addressing mode

1-> 10-bits address mode.

0-> 7-bits address mode.

SMBUS Burst mode: SMBUS burst mode is enabled or not.

1-> burst mode is enabled.

0-> burst mode is disabled.

End SMBUS with ACK: SMBUS end read command with ACK or NACK

1-> send NACK to end.

0-> send ACK to end.

Stop condition (master mode only): send stop condition at the end of transaction.

1-> send stop condition.

0-> no stop condition.

Start condition (master mode only): send start condition at the beginning of transaction.

1-> send start condition.

0-> no start condition.

3.1.2.5 I2C_SA

Slave address: indicates the slave address. If 10 bits mode then, all 9:0 bits used. If 7 bits mode, all 6:0 bits are used.

3.1.2.6 I2C_PRU_PIN

PRU GPO EDIO SDA: indicates the pin number from EDIO to be used as output.

PRU GPI SDA: indicates the pin number for PRU GPI to be used as input.

PRU GPO SCL: indicates the pin number from PRU GPO to be used as CLK.

3.1.2.7 I2C_PRU_CMD_CODE

Command Code: Smbus protocol supports sending a command code in its read/write commands. This 8 bit value indicates the value of command code.

3.1.2.8 I2C_PRU_INST_ID

I2C FW Instance ID: PRU is capable of running upto 4 instance of i2c. This field gives the id to each instance. The id is utilized by driver for identification of interrupt when more than one instance is active.

3.1.2.9 I2C_PRU_TX_DATA

TX Data: This memory buffer is used by I2C firmware for reading the transmitted data. The memory region is 256 Bytes. The amount of data that can be read from the buffer will depend on I2C_BUF register. If buffer is less than 128 bytes, firmware will keep any memory region after buffer as reserved.

3.1.2.10 I2C_PRU_RX_DATA

RX Data: This memory buffer is used by I2C firmware for writing the received data. The memory region is 256 Bytes. The amount of data that can be written in the buffer will depend on I2C_BUF register. If buffer is less than 128 bytes, firmware will keep any memory region after buffer area as reserved.

3.2 Design Challenge

There is one challenge when designing the I2C firmware using PRU core. PRU hardware does not support generic GPIO. It does not have an output enable register, which allows deciding the mode of GPIO pins. They are separate hardware pins. In order to read the value on the pin, GPI has to be used. In order to set a value on the pin, GPO has to be used. To switch between GPI and GPO mode, the pin-mux register of the pin needs to be modified.

There are 3 options in order to resolve this situation. We will be using option 3 for our implementation.

3.2.1 Pinmuxing using PRU.

We can use the PRU core to directly modify the value in pinmux control register. We can then switch between GPO and GPI mode.

Due to hardware limitations, PRUs of AM3 and AM4 are not capable of editing the Pinmux Control registers. At the same time, K2G and AM57xx has a required procedure for editing pinmux registers in order to ensure the IO timings in the SOC data manual over the lifetime of the device. According to this procedure, the PRU should not edit the pinmux registers on the fly.

3.2.2 Pinmuxing using EDMA.

We can use the EDMA engine to modify the value in pinmux control register. The following timing diagram explains the situation.

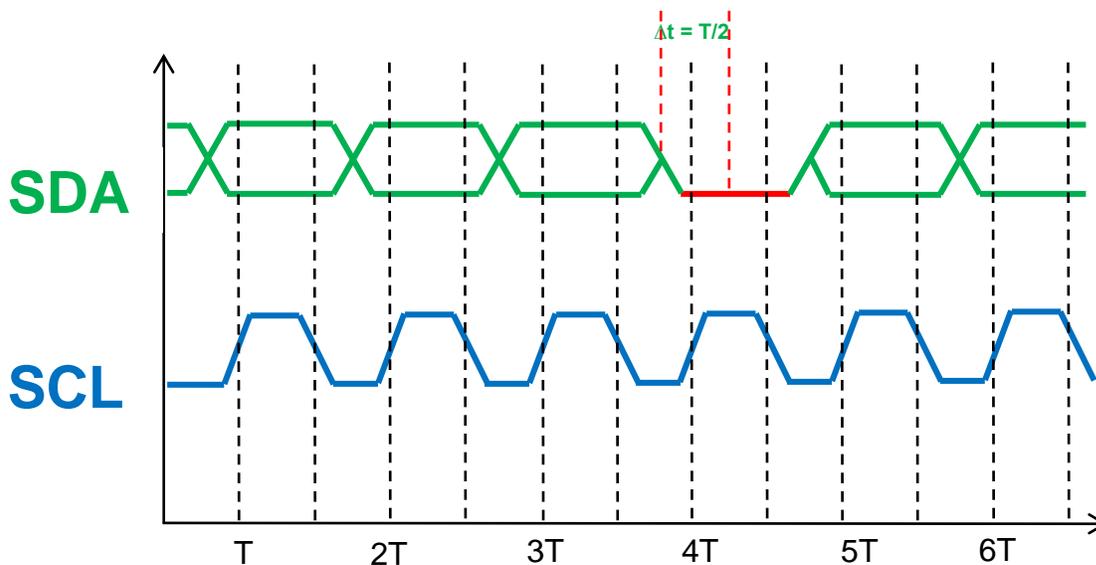


Figure 2. pinmux switching timing diagram

The following are the timing requirement need by the I2C protocol. Here, Δt is the time period in which the GPI/GPO switching needs to be done.

HS I2C i.e. I2C at 400 KHz,	Time T = 1/400 KHz = 2.5 uSec
	$\Delta t = T/2 = 1.25$ uSec
Standard I2C i.e. I2C at 100 KHz,	Time T = 1/100 KHz = 10 uSec
	$\Delta t = T/2 = 5$ uSec

EDMA time for writing one pinmux control register $t \approx 3.3$ uSec.

Therefore, we cannot proceed with this option if we want to support HS mode in I2C firmware.

3.2.3 Using IEP DIGIO pins.

ICSS subsystem comes with an Industrial Ethernet Peripheral. IEP comes with a digital I/O port (DIGIO). One important feature for IEP DIGIO is that it can support tristate mode. It means the GPO support 3 logic level value high, low and tristate. This makes it possible to emulate Hardware GPIO IP feature using PRU GPI and GPO pins.

2 pins of PRU are used to emulate the SDA line for I2C. They are DIGIO GPO pin and PRU GPI pin. The following connection diagram shows it is done.

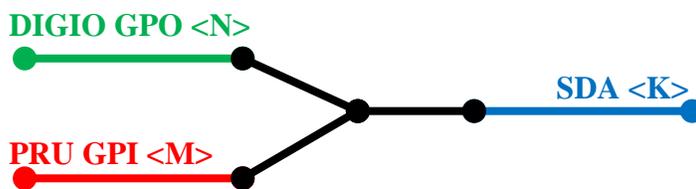


Figure 3. Electric connection for SDA line

In order to set a value high or low on the SDA line the DIGIO GPO pin is set to high or low value. In order to read the pin value on SDA line, the DIGIO GPO pin is set to tristate and then PRU GPI pins read the value on the SDA line.

There are some limitations to this method too.

- 1) There are only 8 DIGIO GPO pins pinned out of SOC (AM3,4,5 & K2G). This makes the absolute maximum number of I2C instance supported per ICSS subsystem to 8 instances.
- 2) For AM3 and AM4, there is no IEP peripheral for ICSS0 subsystem. Only ICSS1 subsystem can ran I2C firmware.
- 3) There is latency in the read and write cycle time to DIGIO GPO. It takes 7 cycles for a write to be propagated on DIGIO output pins. This makes read cycle time on the GPI pins to 8 cycles (7 for writing DIGIO GPO to tristate and 1 for reading PRU GPI pins).

3.3 PRU Resource Usage

3.3.1 PRU Data RAM

The firmware design uses PRU DATA RAM i.e. DRAM for creating the Memory Map register section. The whole DRAM is divided into following section.

3.3.1.1 Configuration Memory Region

This memory region is used for giving global configuration for Firmware. Currently, this region has 3 registers. The following are descriptions of those registers.

Register name	Offset	TYPE	Bits	Description
IEP_INIT_COUNT	0x00	RW	64:0	This value indicates the first count value for IEP compare event. This register is used for event when IEP peripheral is already running.
IRQ_STATUS_REG	0x08	RW	3:0	This is one common IRQ register for all instances. This allows for a quick look up during ISR routine.
BUS_FREQUENCY	0x0C	RW	3:0	This decides the bus frequency I2C Firmware. The Firmware decides the routine based on this register.

Table 7. Configuration Memory Map

3.3.1.2 Instance Memory Region

The description of this memory regions is mentioned [Section 3.1](#). These can maximum support upto 4 Instance memory.

- a. I2C0 Memory region
- b. I2C1 Memory region
- c. I2C2 Memory region
- d. I2C3 Memory region

3.3.2 Local Register

PRU core have 32 local CPU registers. These registers are 1, 2 and 4 bytes addressable. The firmware design uses each register for storing different firmware related information. The following table shows the list of register usage as well as the data stored.

Register	Bits	Description
R10	31:0	This register is used to keep a pointer to Instruction Memory region for quick access.
R11	31:0	This register is used to keep a pointer to Tx Memory Buffer for quick access.
R12	31:0	This register is used to keep a pointer to Rx Memory Buffer for quick access.
R13	31:16	The upper 16 bits is used to store the slave address for current transaction.
	15:0	The lower 16 bits is used to store the state pointer of the current instance. Hence, this helps firmware to identify the action to perform and next state.
R14	31:24	Used for storing instance id of current instance.

	23:16	Used for storing pin number for EDIO to be used as output.
	15:8	Used for storing pin number for PRU GPI to be used as input.
	7:0	Used for storing pin number for PRU GPO to be used as CLK.
R15	31:24	Used for storing total data count for current transaction.
	23:16	Used for storing current data sent/received for current transaction.
	15:8	Used for storing data value sent/received for current transaction.
	7:0	Used for storing number of Address/Data bits sent/received.
R16	31:24	Used for storing Rx buffer size.
	23:16	Used for storing Tx buffer size.
	15	Used for storing I2C module enable bit for current transaction.
	10	Used for storing Master/slave mode bit for current transaction.
	8	Used for storing Expand Slave address bit for current transaction.
	5	Used for storing SMBUS Burst mode bit for current transaction.
	4	Used for storing End SMBUS with ACK bit for current transaction.
	1	Used for storing Stop condition bit for current transaction.
	0	Used for storing Start condition bit for current transaction.
R17	31:16	Used for SMBUS DATA count.
	15:0	Used for storing global state pointer for current instance.
R18	31:0	Used for storing IEP DIGIO output enable register's local copy for PRU0.
R19	31:0	Used for storing IEP DIGIO output enable register's local copy for PRU1.

Table 8. CPU Register Usage

3.3.3 Scratchpad

There are 3 shareable scratchpad memory banks available in PRU ICSS. They are used for storing active state of each instance at given time. The firmware pushes all the data from PRU Core registers into these memories depending on PRU core and instance number and vice versa. The following table indicates usage of memory banks.

PRU CORE	Instance	Bank	Registers
PRU0	I2C0	BANK0	REG 0 to REG 9
	I2C1		REG 10 to REG 19
	I2C2		REG 20 to REG 29
	I2C3	BANK1	REG 0 to REG 9
PRU1	I2C0	BANK2	REG 0 to REG 9
	I2C1		REG 10 to REG 19
	I2C2		REG 20 to REG 29
	I2C3	BANK1	REG 20 to REG 29

Table 9. Scratchpad Memory Usage

3.4 Design Theory

3.4.1 Initialization Task

The firmware logic, when it comes out of reset does all the required initialization of various peripheral and component needed for functioning. The resources it initializes are IEP Timer and Compare events, IEP DIGIO logic level setup, enable support for using scratchpad memory. The following state flow diagram shows the steps done by firmware during initialization task.

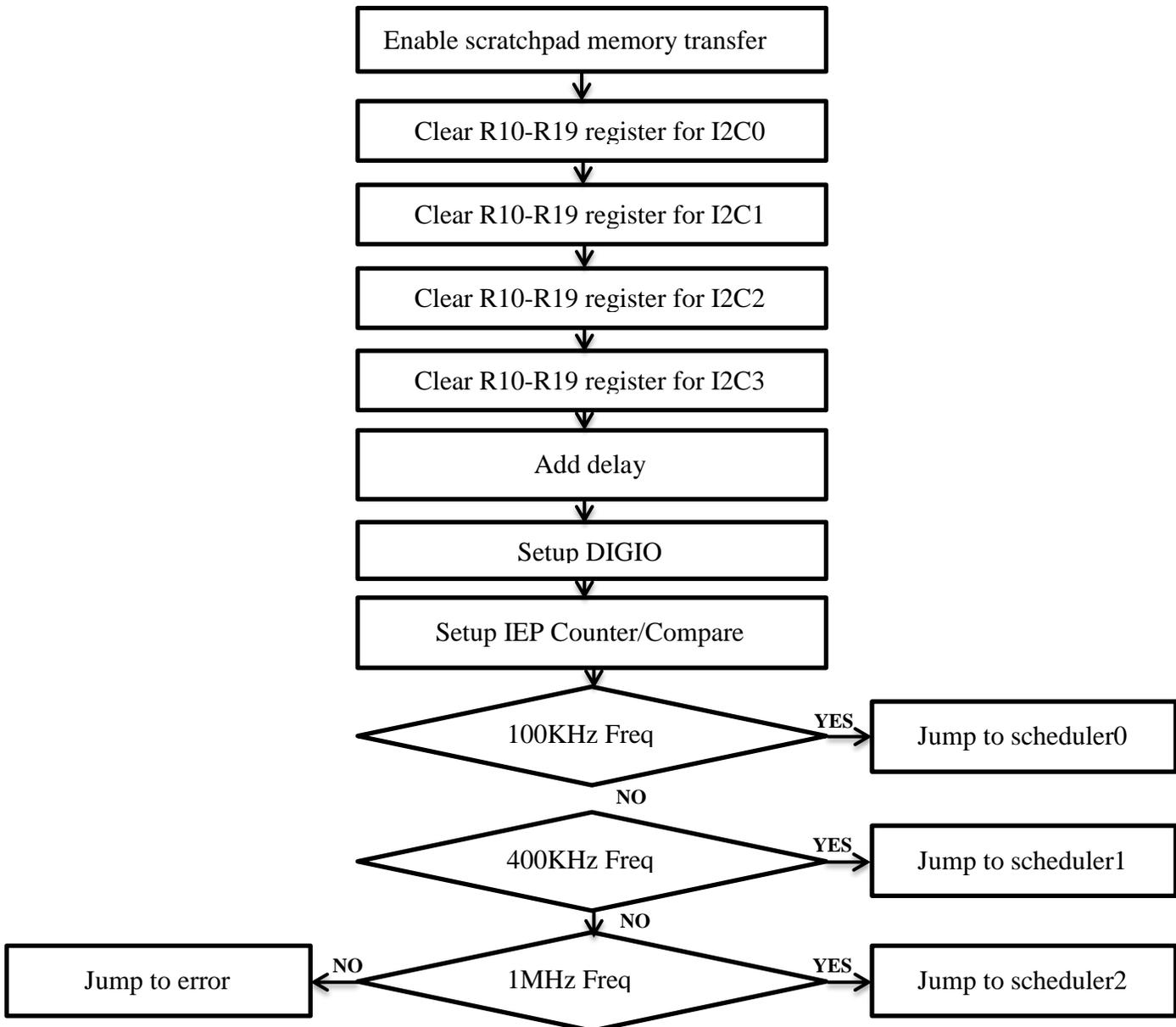


Figure 4. Init Task

The first things firmware does after coming out of reset is that it enable support of transferring data between scratchpad memory bank and local CPU registers. It initializes R10-R19 registers for instance 0 and set the state pointer to RESET state and then store the value of register into memory banks. It does the same for remaining instances. Then it enables IEP DIGIO register to low logic value and enable to pins to go to high impedance state. Finally, it set the IEP compare event based on the bus frequency provided.

3.4.2 Scheduler Task

The firmware would be able to emulate multiple independent instances of I2C firmware across different pins. Example: It can emulate I2C instance 0 on GPIO pin0 and pin1. It can emulate I2C instance 1 on GPIO pin2 and pin3. Both instances will be working independently from each other irrespective of each state. In order to do this, it requires a scheduler task.

The purpose of the scheduler task is to switch between each instance at regular interval of time. Every instance is provided a time slot and it should fulfill its activity during the time slot. Whenever it receives the interrupt for IEP timer, it means the interval for a particular instance is over, and it will move to next instance.

There are limitations on PRU resources based on HW. Hence, there are 3 version of scheduler. One for 100KHz Bus speed, one for 400KHz and one for 1MHz.

The overall state diagram of the scheduler is as follows.

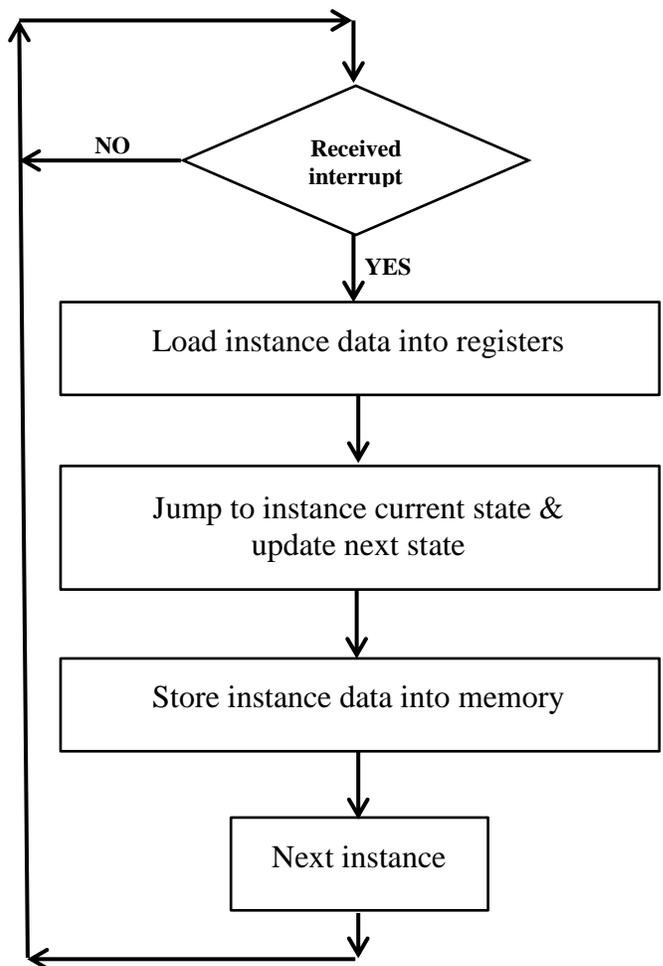


Figure 5. scheduler state function

The state diagram starts with reading the interrupt pin if the interrupt has been raised or not. Once, it receives the interrupt then first it calculates the next instance. There are multiple of factors affecting the outcome. Example, how many instances are enabled currently. After that, the working meta data for that instance is loaded back into the registers. Based on the meta data, the current state is found out and next state is calculated. Once the operation is performed, the data is stored back into the memory. Here memory can be data memory or scratch pad memory.

Note: meta data consist of state information, pointers to data memory, data value, address value etc.

3.4.3 Communication with host

Contrary to usual hardware IP registers, firmware cannot understand if any register has been updated or not. Therefore in order to update the configuration of firmware, we need to use a communication protocol between Host and firmware running PRU core. The following describes how that communication between host and firmware is done.

There is a register provided in the firmware register map of I2C firmware. It has been named as **I2C_COMMAND**. This register is used to communicate between host and pru core. There are 2 16 bits field in the register “**Command Word**” & “**Command Response**”. Both core has to agree on the following policy.

- 1) Host will update **Command Word** if the current value of the field is 0x0000. Hence, it will first read the value and if it is 0x0000 then only it will update the value.
- 2) Host will only write 0x0000 in **Command Response**. It can read the value any time but only write 0x0000 to the field.
- 3) PRU will update **Command Response** if the current value of the field is 0x0000. Hence, it will first read the value and if it is 0x0000 then only it will update the value.
- 4) PRU will only write 0x0000 in **Command Word**. It can read the value any time but only write 0x0000 to the field.

The typical scenario of how the communication that will happen between Host and PRU is shown below.

- 1) Host and PRU are active. Host will update the command word to command code i.e. 0x#### and command response to 0x0000. After that I will go to sleep.

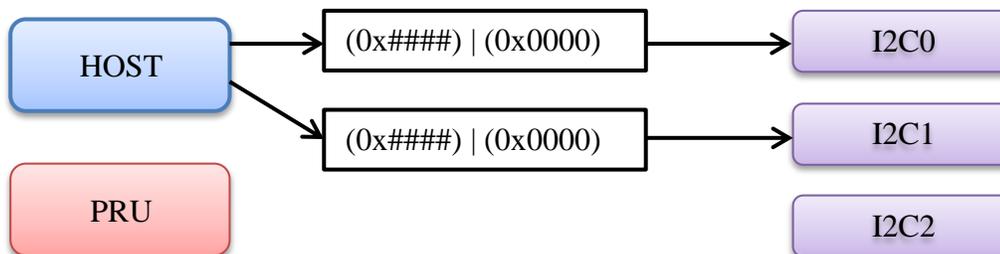


Figure 6. HOST sends command to PRU

- 2) PRU will constantly read the command word and command response value. Once response value becomes 0x0000, it will read and perform the action indicated by the command word.

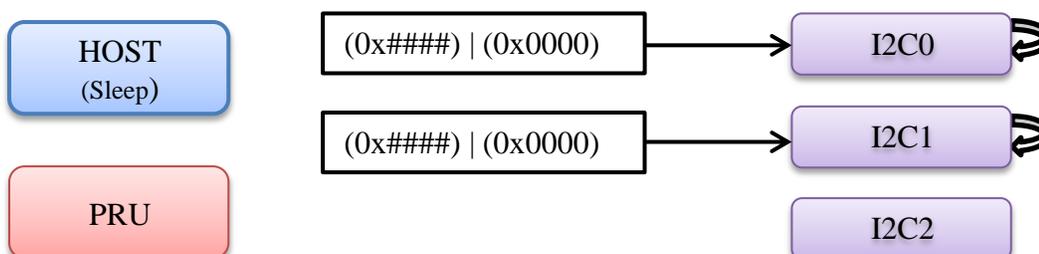


Figure 7. PRU reads command from HOST

- PRU will finish the action with success or failure. It will update the response register accordingly. It will update command word to zero indicating that command is finished. Finally, it will raise an interrupt via INTC register.

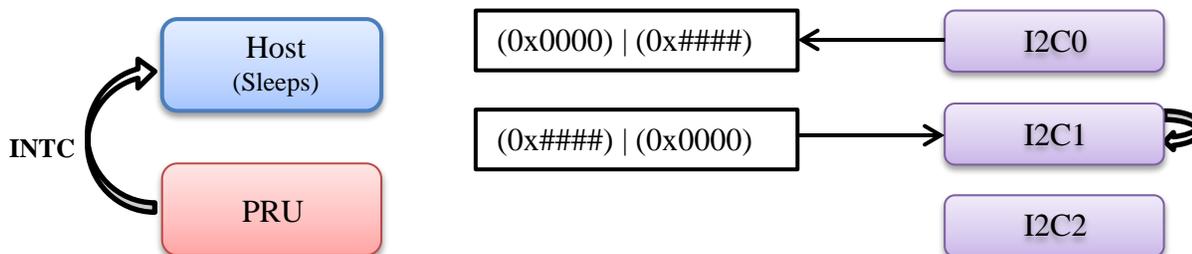


Figure 8. PRU responds back to HOST

- Host will receive the interrupt. In the ISR routine, Host will read the command word and response for all instances. Host will find which instances responded. It will check if the command word is 0x0000 and response is anything except 0x0000. It will take appropriate action based on response value.

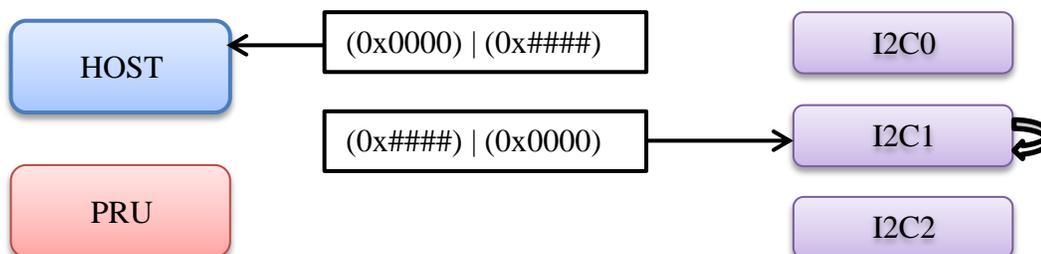


Figure 9. Host reads response from PRU

Both Host and PRU will follow the policy. This will make sure that no command is lost due to any timing mismatch. Once a command is passed, host cannot do anything until PRU responds to the command. Similarly, PRU cannot do anything until the host has accepted the response.

3.4.3.1 Interrupt support for Host

PRU hardware supports limited number of interrupt available for host. Firmware uses one interrupt line between host and PRU core for providing interrupt calls. This interrupt line is shared between all instances. The usual way for PRU to raise interrupt is in response to any command sent.

In order to provide quick ISR response, the [IRQ_STATUS_REG](#) is used. Each will update the bit field in this register based on the id provided to that instance. Using the register and id host will realize which interrupt has been raised. For example, instance 0 will make bit 0 high and it will cleared by the host when interrupt is served. Similarly, instance 1 will make bit 1 high and it will cleared by the host when interrupt is served. The order of serving of interrupt will not matter, as if the interrupt is not served by the host in a fixed interval, the firmware will try to fire the interrupt until it has been served.

The following are the sequence of steps done by firmware for raising interrupt for each instance.

- 1) **The firmware will update I2C1 command response.**
- 2) **The firmware will set the bit field high in [IRQ STATUS REG](#) register.**
- 3) **Assert the PRU INTC IRQ line.**
- 4) **Wait/check for bit field to be cleared to zero.**
- 5) **Keep on asserting the IRQ line every alternate I2C clk cycle until the bit field is cleared to zero.**
- 6) **Once cleared move to next state.**

The following are the sequence of steps expected from Host for serving the interrupt.

- 1) **Disable the IRQ.**
- 2) **Read the bit field in [IRQ STATUS REG](#) register and find out how many and which PRUs has raised the interrupt.**
- 3) **Clear the bit field to zeros.**
- 4) **De-assert the PRU INTC IRQ line.**
- 5) **Exit IRQ handler and re-enable the IRQ.**

3.4.4 I2C protocol states

The primary working state of i2c firmware is as follows. This is the generalized state of each i2c protocol. Each instance has a copy of its own state.

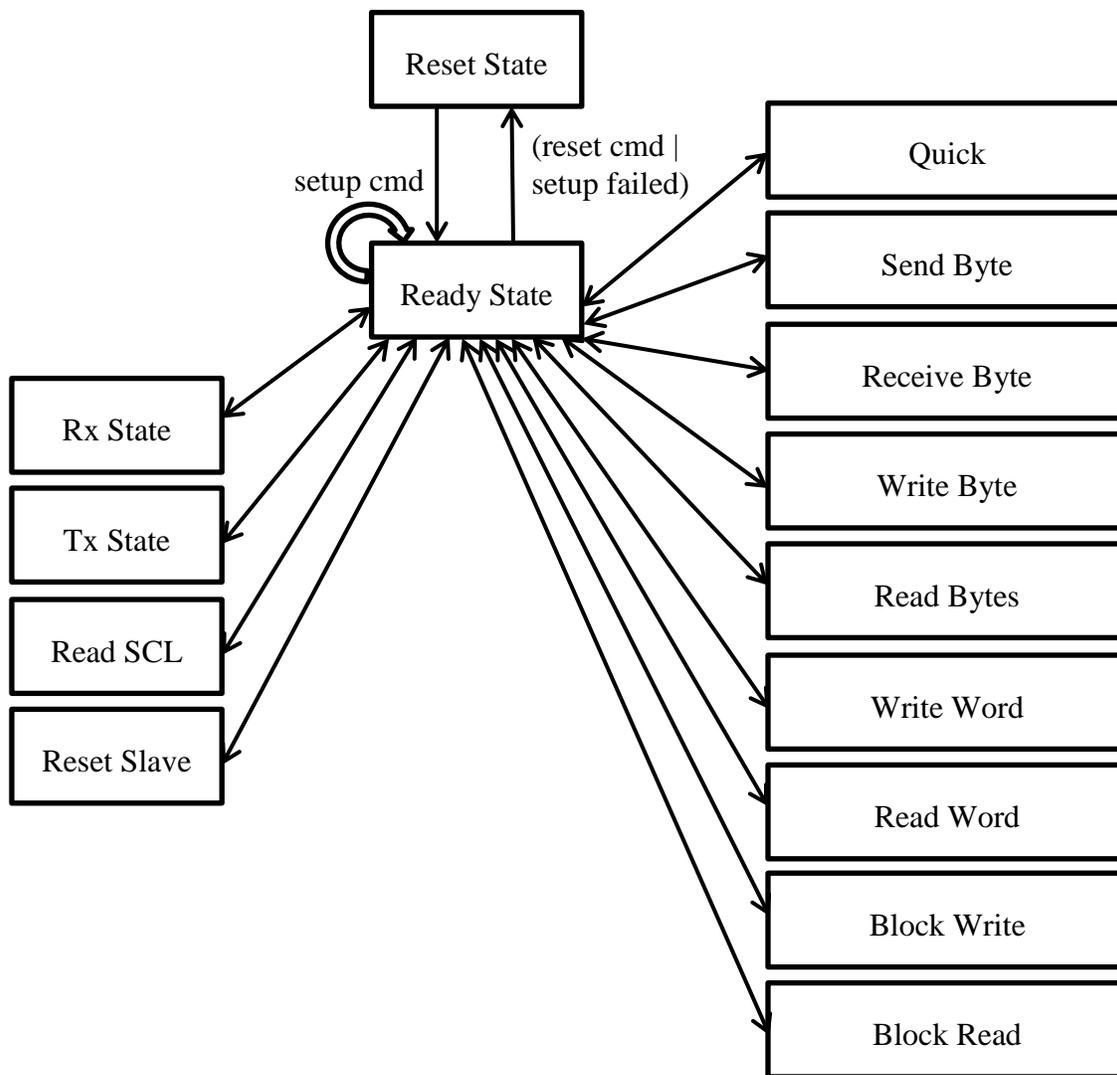


Figure 10. Global State Diagram

There are 15 primary states. Reset state is the default state when the system comes up. It does not have any configuration at this time. Once, setup command is received, it does all the parameter configuration and moves to ready state. In order to change parameter, pass the setup command again with different parameter values. The state changes to Rx state on receiving a receive command. It changes to Tx state on receiving a transmit command. Similarly, it changes to Quick state on receiving an SMBUS quick command. If a failure happens it moves back ready state. For example, any failure with block read/write or byte read/write state etc. it will move the state back to ready state. Each of this states are divided into multiple stages.

3.4.4.1 Reset

In this state, firmware looks for 2 things. If the module is enabled and setup command received then it moves to ready state.

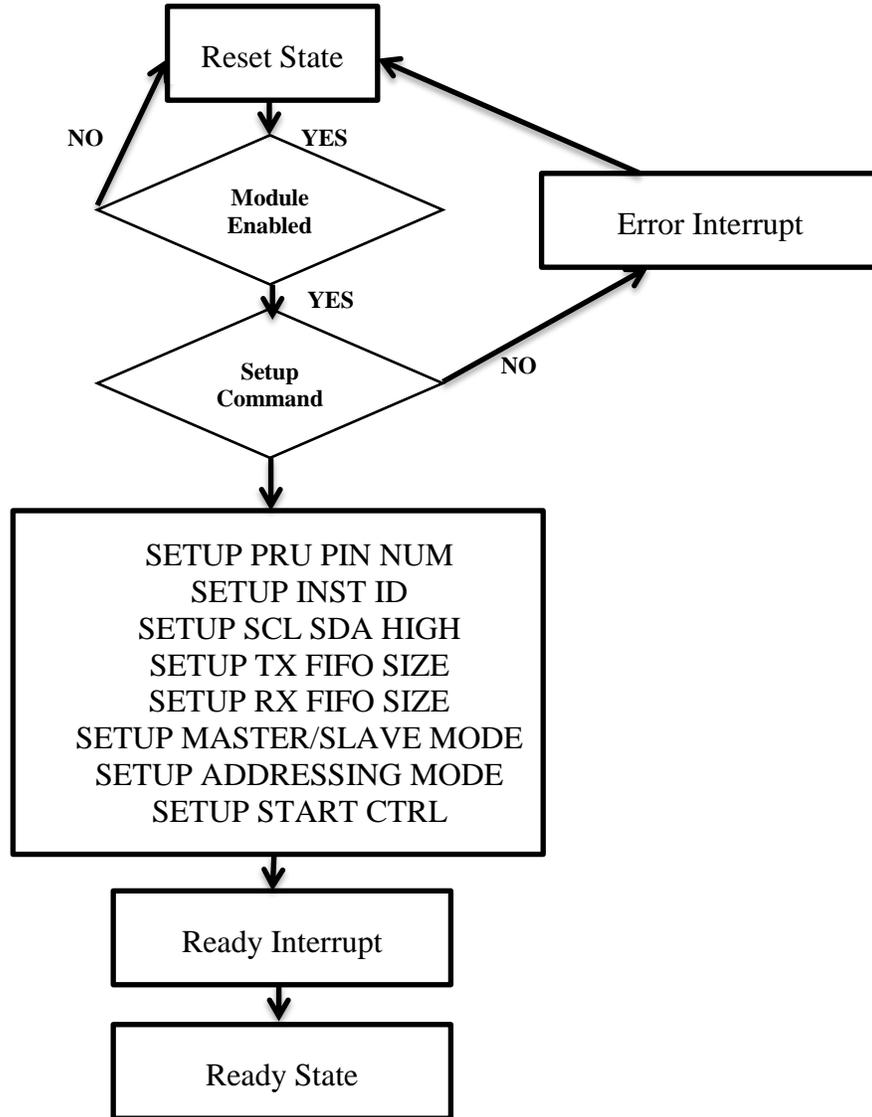


Figure 11. RESET State

3.4.4.2 READY

The firmware in this state goes wait for command to be passed. Once, receiving a command it goes to next state or gives an error interrupt for unknown command.

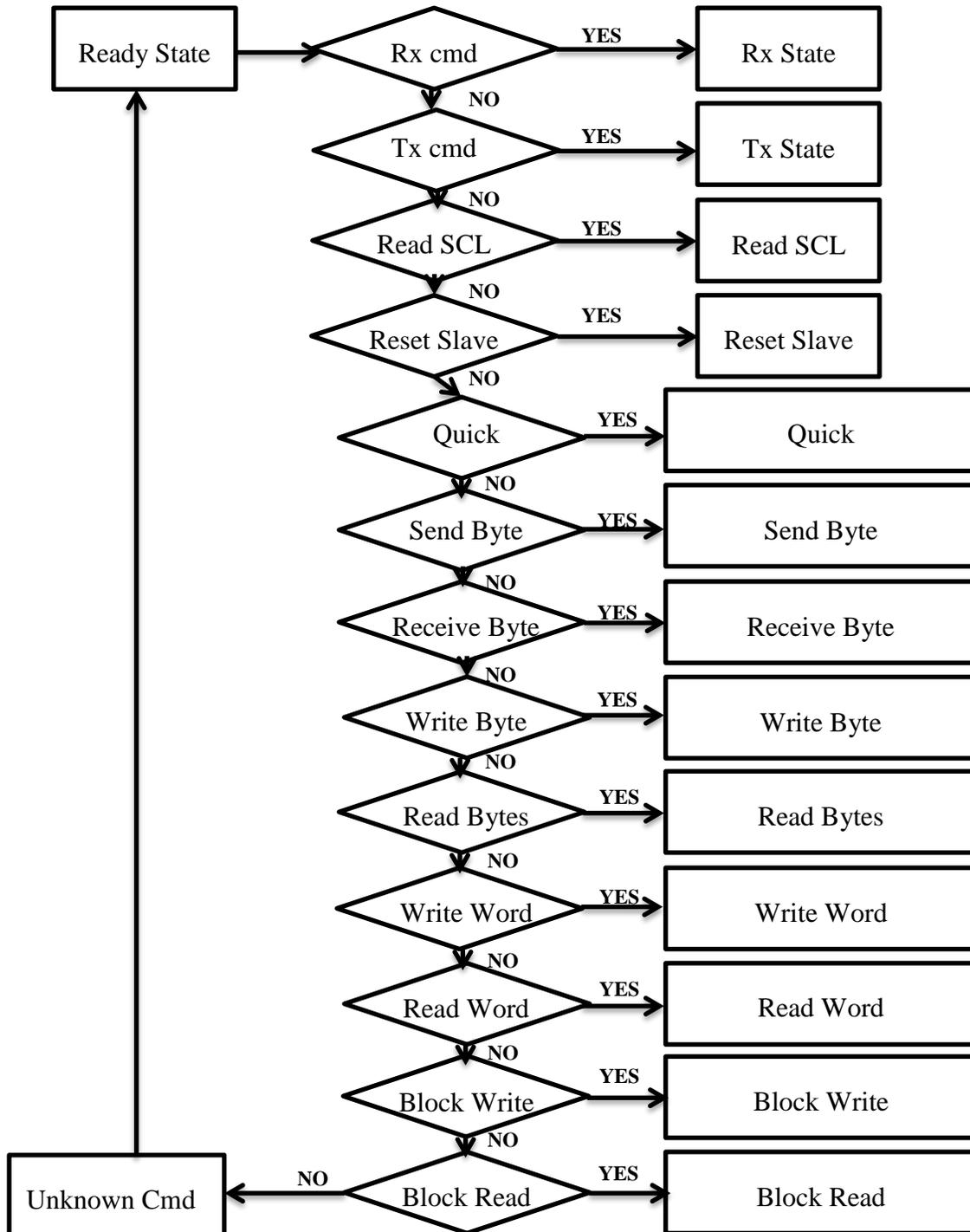


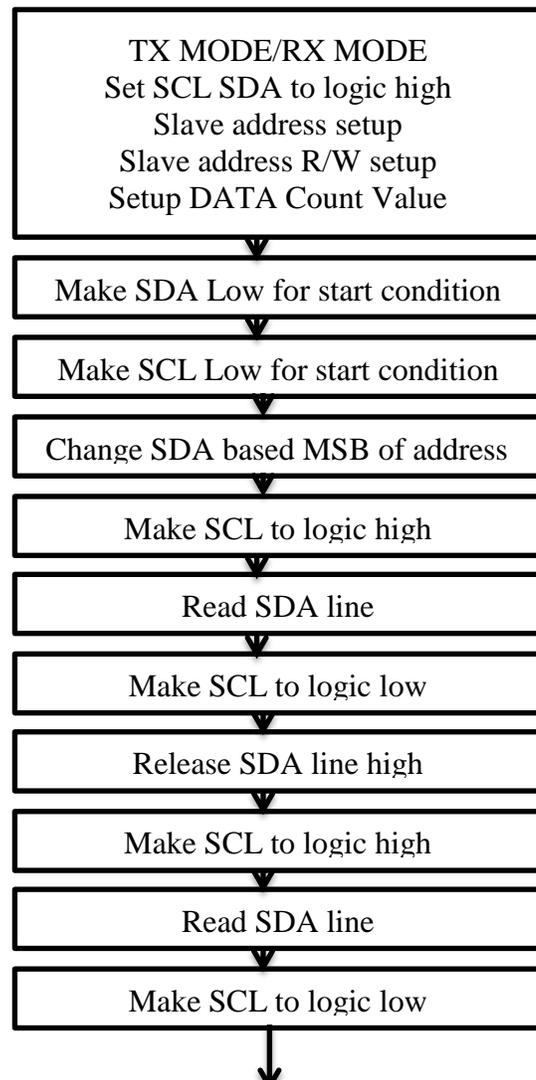
Figure 12. Ready state transtion

3.4.4.3 DATA Transfer

The following are the command which makes the firmware goes into data transfer state.
The following are the list of state in the broad state.

- 1) Rx: For Standard I2C receive data
- 2) Tx: For Standard I2C send data
- 3) Quick: For SMBUS quick command
- 4) Send byte: For SMBUS sending 1 byte
- 5) Recieve byte: For SMBUS receiving 1 byte
- 6) Write byte: For SMBUS writing 1 byte
- 7) Read byte: For SMBUS reading 1 byte
- 8) Write word: For SMBUS writing 2 byte
- 9) Read word: For SMBUS read 2 byte
- 10) Block write: For SMBUS writing N bytes
- 11) Block read: For SMBUS read N bytes

The general flow in these states is as follows.



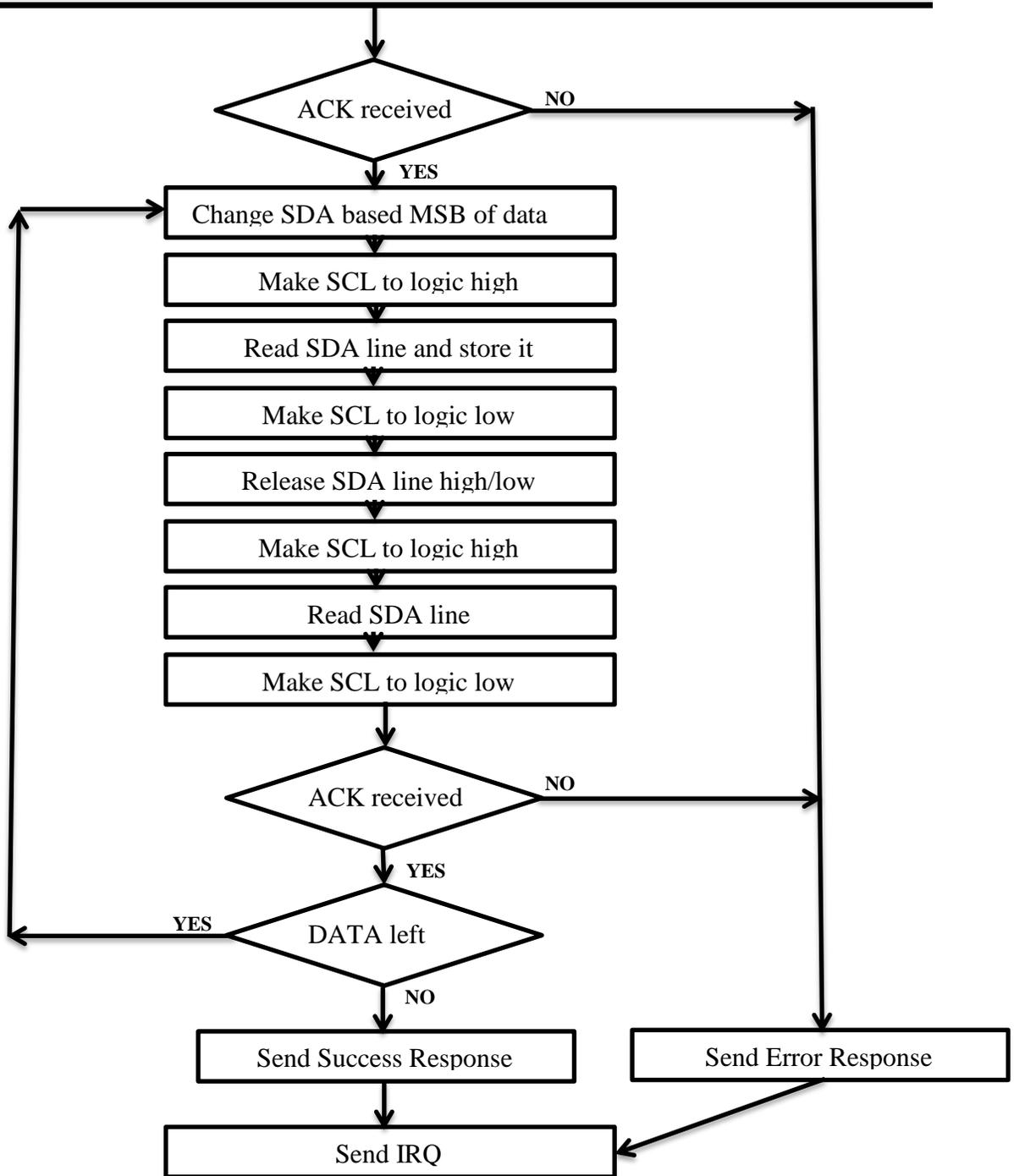


Figure 13. General Data Transfer Flow

3.4.4.4 Read SCL

The purpose of this command is to read the SCL line in case the slave is hung or something bad has happened. In this state, the firmware reads the SCL line for 10 I2C clock cycles. It responds with high or low value. If the firmware reads the SCL line is pulled down for all 10 clock cycles, then it responds with low value else with high value. The following diagram shows the state flow.

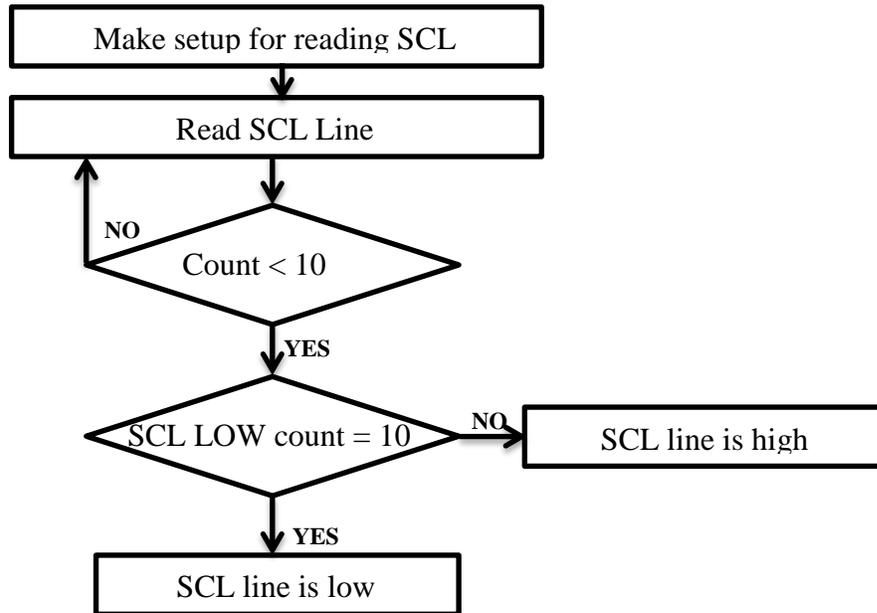


Figure 14. Read SCL Line

Note: In order to read the SCL line its pinmux register has to be modified from GPO to GPI. PRU is not capable for changing the SCL pinmux. The host has to do this task for firmware before passing this command.

3.4.4.5 Reset Slave

This command is used of resetting the slave using standard 9 clock cycles. If the slave is hung this is the only way to bring it out of reset.

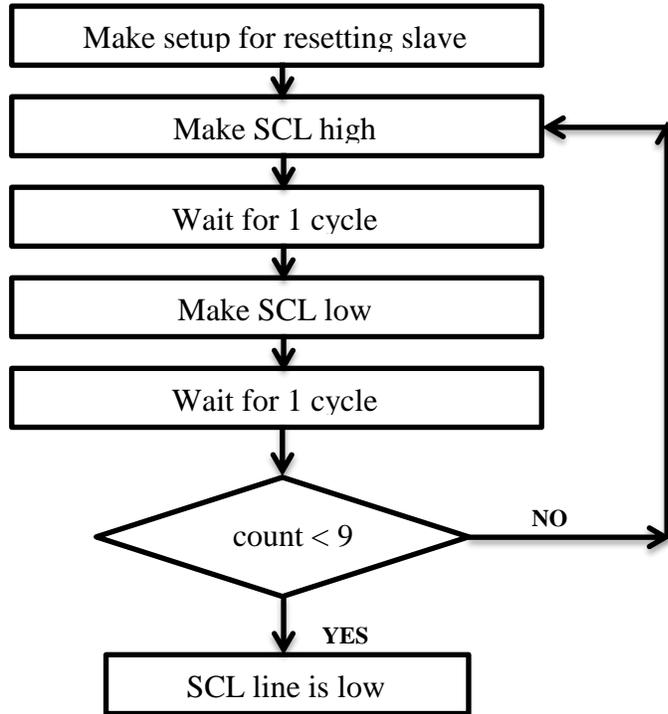


Figure 15. RESET slave

3.4.5 Concurrent execution

In order to truly emulate independent multiple instance of i2c protocol, we break the processing time of each instance into small time interval. The following time graph will show how it is done.

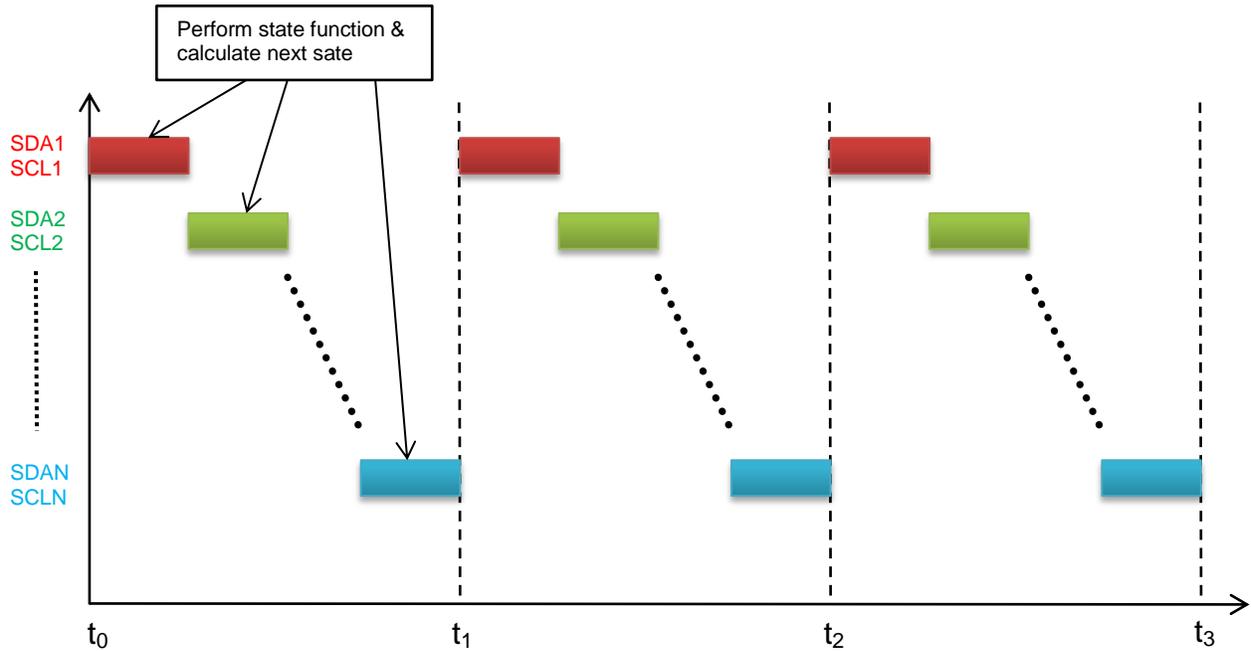


Figure 16. Scheduling graph for firmware

3.5 Firmware Source code

3.5.1 Firmware Macros Description

The following are the list of macros used in the firmware source code.

Macros	File	Function
UPDATE_NEXT_LOCAL_STATE	I2C_macros.h	update next state in state keep register.
UPDATE_NEXT_GLOBAL_STATE	I2C_macros.h	update next state in state keep register.
COPY_LOCAL_TO_GLOBAL_STATE	I2C_macros.h	update next state in state keep register.
STATE_TASK_OVER	I2C_macros.h	Return to the scheduler as state task is over.
CHECK_INTERRUPT_RECEIVED	I2C_macros.h	Check if the interrupt is recieved by the host & then move to next task.
RAISE_INTERRUPT_MEM_FOR_HOST	I2C_macros.h	raise the interrupt memory for telling host which instance raise the interrupt.
RAISE_INTERRUPT_FOR_HOST	I2C_macros.h	raise the interrupt line for Host.
SET_EDIO_DATAOUT_VALUE	I2C_macros.h	Set value on all iep digio pins
SET_SDA_PIN_LOW	I2C_macros.h	Set low value on SDA pin
SET_SDA_PIN_HIGH	I2C_macros.h	Set high value on SDA pin
SET_SCL_PIN_LOW	I2C_macros.h	Set low value on SCL pin
SET_SCL_PIN_HIGH	I2C_macros.h	Set high value on SCL pin
READ_SDA_PIN_ACK	I2C_macros.h	Set high value on SCL pin
SET_OUTPUT_PIN_VALUE_HIGH	I2C_macros.h	Set high value on SCL and SDA pin
READ_ADDRESS_REGISTER	I2C_macros.h	Read the address register for slave address
READ_RW_REGISTER_BIT	I2C_macros.h	Read the RW bit to find read or write operation
SET_SDA_LOW_FOR_START	I2C_macros.h	set sda low for start condition
SET_SCL_LOW_FOR_START	I2C_macros.h	set scl low for start condition
MODIFY_SDA_PIN	I2C_macros.h	Change sda pin based on the data register
READ_SDA_PIN	I2C_macros.h	read sda pin based on the line value
SCL_LOW_NEXT_STATE	I2C_macros.h	Change scl to low and decide next state
MOV32	I2C_scheduler.h	Move a 32bit value to a register
DELAY	I2C_scheduler.h	ADD a delay of N cycles.
ENABLE_XIN_XOUT_SHITFTING	I2C_scheduler.h	ADD a delay of N cycles.
I2C_SETUP_IEP_COUNTER	I2C_scheduler.h	Setup the IEP timer counter for periodic interrupt.
I2C_SETUP_IEP_DIGIO	I2C_scheduler.h	Setup the IEP timer counter for periodic interrupt.
I2C_INSTANCE0_INIT	I2C_scheduler.h	Initialize all the register for I2C0
I2C_INSTANCE1_INIT	I2C_scheduler.h	Initialize all the register for I2C1
I2C_INSTANCE2_INIT	I2C_scheduler.h	Initialize all the register for I2C2
I2C_INSTANCE3_INIT	I2C_scheduler.h	Initialize all the register for I2C3
I2C_IEP_INTC_CLEAR_EVENT	I2C_scheduler.h	Clear the iep cmp event and intc event for Standard/Full mode

I2C_IEP_INTC_CLEAR_EVENT1	I2C_scheduler.h	Clear the iep cmp event and intc event for HS mode
I2C_WAIT_FOR_IEP_CMP	I2C_scheduler.h	wait until the IEP CMP Event triggers and interrupt
I2C_WAVE_FUNCTION0	I2C_scheduler.h	jump to the next state of i2c0 function
I2C_WAVE_FUNCTION1	I2C_scheduler.h	jump to the next state of i2c1 function
I2C_WAVE_FUNCTION2	I2C_scheduler.h	jump to the next state of i2c2 function
I2C_WAVE_FUNCTION3	I2C_scheduler.h	jump to the next state of i2c3 function

Table 10. Macros List

3.5.2 Firmware Sources Description

The following are the list of firmware source code files.

File	Description
firmware_version.h	Contains the version information of the firmware.
I2C_function.h	Contains the function routine of Rx and Tx part of I2C protocol
I2C_macros.h	Contains all the macros needed for Rx/Tx for I2C protocol
I2C_protocol.asm	Contains the routine for global state function
I2C_scheduler.asm	Contains the scheduler routine for multiple instances
I2C_scheduler.h	Contains all the macros needed scheduler routine
I2C_smbus.asm	Contains the routine needed for SMBUS operation
icss_cfg_regs.h	Contains ICSS Configuration Register definition
icss_defines.h	Contains ICSS Global Defines
icss_i2c.h	Contains I2C firmware related Offset and Register definition
icss_iep_regs.h	Contains ICSS Industrial Ethernet Peripheral Registers Definition
icss_intc_regs.h	Contains ICSS Interrupt Controller Module Registers Definition
icss_miirt_regs.h	Contains ICSS MII_RT Module Registers Definition
pru.cmd	Linker cmd file of firmware builds

Table 11. Firmware Source List

4 RTOS Driver Support

4.1 External APIs

Currently Hard IP driver supports a list of APIs for configuring I2C IP. This APIs are called by application to configure and used I2C IP. The same API's will be supported for I2C firmware also. Application will call the APIs in the same manner for using I2C firmware. The following are the list of API's.

Return Type	API name	Arguments	Functional Description
void	I2C_close	handle	A I2C_Handle returned from I2C_open
int32_t	I2C_control	handle	A I2C handle returned from I2C_open()
		uint32_t	A command value defined by the driver specific implementation
		void*	An optional R/W (read/write) argument that is accompanied with cmd
void	I2C_init	void	The I2C_config structure must exist and be persistent before this function can be called
I2C_Handle	I2C_open	uint32_t	Logical peripheral number for the I2C indexed into the I2C_config table
		I2C_Params*	Pointer to an parameter block, if NULL it will use default values. All the fields in this structure are RO (read-only).
void	I2C_Params_init	I2C_Params*	An pointer to I2C_Params structure for initialization
int16_t	I2C_transfer	I2C_Handle	A I2C_Handle returned from I2C_open
		I2C_Transaction*	A pointer to a I2C_Transaction. All of the fields within transaction are WO (write-only) unless otherwise noted in the driver implementations
void	I2C_transactionInit	I2C_Transaction*	transaction parameter structure to initialize

Table 12. List of Application APIs

API name	Functional Description
I2C_close	Function to close a I2C peripheral specified by the I2C handle
I2C_control	Function performs implementation specific features on a given I2C_Handle
I2C_init	Function to initializes the I2C module
I2C_open	Function to initialize a given I2C peripheral specified by the particular index value. The parameter specifies which mode the I2C will operate.
I2C_Params_init	Function to initialize the I2C_Params struct to its defaults
I2C_transfer	Function that handles the I2C transfer
I2C_transactionInit	Function to initialize the I2C_Transaction struct to its defaults

Table 13. Function Description of APIs

4.2 Internal Files

New version of source file will be implemented in I2C_v2.c. This version of the file will implement the APIs in case of firmware Soft IP. There are lot of internal function which is driver implementation dependent.

External API name	Mapped internal implementation
I2C_close	I2C_close_v2
I2C_control	I2C_control_v2
I2C_init	I2C_init_v2
I2C_open	I2C_open_v2
I2C_Params_init	I2C_Params_init_v2
I2C_transfer	I2C_transfer_v2
I2C_transactionInit	I2C_transactionInit_v2
-	I2C_v2_pruIcssInit
-	I2C_v2_setupPinMux
-	I2C_v2_writePinNum
-	I2C_v2_setupFifoSize
-	I2C_v2_pollIrqSts
-	I2C_v2_setBusFrequency
-	I2C_v2_enableModule
-	I2C_v2_disableModule
-	I2C_v2_enableMasterMode
-	I2C_v2_enableSlaveMode
-	I2C_v2_7bitAddressMode
-	I2C_v2_10bitAddressMode
-	I2C_v2_enableStartBit
-	I2C_v2_disableStartBit
-	I2C_v2_enableStopBit
-	I2C_v2_disableStopBit
-	I2C_v2_sendCmd2PRU
-	I2C_v2_readResp2PRU
-	I2C_v2_clearResp2PRU
-	I2C_v2_setSlaveAddress
-	I2C_v2_putSmbusCmdCode
-	I2C_v2_setDataCount
-	I2C_v2_getDataCount
-	I2C_v2_putData
-	I2C_v2_getData
-	I2C_v2_waitForCompletion
-	I2C_v2_sendNack
-	I2C_v2_sendAck
-	I2C_v2_enableBurstMode
-	I2C_v2_disableBurstMode

-	I2C_v2_writeInstId
---	--------------------

Table 14. Firmware Internal APIs

5 Test Plans

5.1 EVM Support

5.1.1 Icev2AM335x

ICSS	PRU	Instance	Functional Pin	PRU GPIO Pins	EVM Port	EVM pin
ICSS1	PRU0	I2C0	SCL	pr1_pru0_pru_r30_1	J3	14
			SDA	pr1_edio_data_out7	J4	21
				pr1_pru0_pru_r31_0	J3	12

Table 15. iceAM335x I2C Instances

5.1.2 idkAM437x

ICSS	PRU	Instance	Functional Pin	PRU GPIO Pins	EVM Port	EVM pin
ICSS1	PRU0	I2C0	SCL	pr1_pru0_pru_r30_8	J3	6
			SDA	pr1_edio_data_out0	J3	5
				pr1_pru0_pru_r31_9	J3	8
		I2C1	SCL	pr1_pru0_pru_r30_10	J16	46
			SDA	pr1_edio_data_out1	J3	7
				pr1_pru0_pru_r31_11	J16	48

Table 16. idkAM437x I2C Instances

5.1.3 idkAM572x

ICSS	PRU	Instance	Functional Pin	PRU GPIO Pins	EVM Port	EVM pin
ICSS1	PRU1	I2C0	SCL	pr1_pru1_gpo1	J21	5
			SDA	pr1_edio_data_out1	J46	4
				pr1_pru1_gpi0	J21	3

Table 17. idkAM572x I2C Instances

5.1.4 idkAM574x

ICSS	PRU	Instance	Functional Pin	PRU GPIO Pins	EVM Port	EVM pin
ICSS1	PRU1	I2C0	SCL	pr1_pru1_gpo1	J21	5
			SDA	pr1_edio_data_out1	J46	4
				pr1_pru1_gpi0	J21	3

Table 18. idkAM574x I2C Instances

5.1.5 idkAM571x

- No PRU pin available for idkAM571x.
 - All PRU GPIO pins are being routed to external ICs.

5.1.6 iceK2G

- No PRU pin available for iceK2G
 - All PRU GPIO pins are being routed to extension connector J4.
 - No daughter card available to connect with the port.

5.2 External I2C board

5.2.1 I2C EEPROM Board

The following are the pictures of the I2C EEPROM board. This board is designed manually for the purpose of testing firmware. It is custom made board. The board comprises of 2 I2C EEPROM chips.

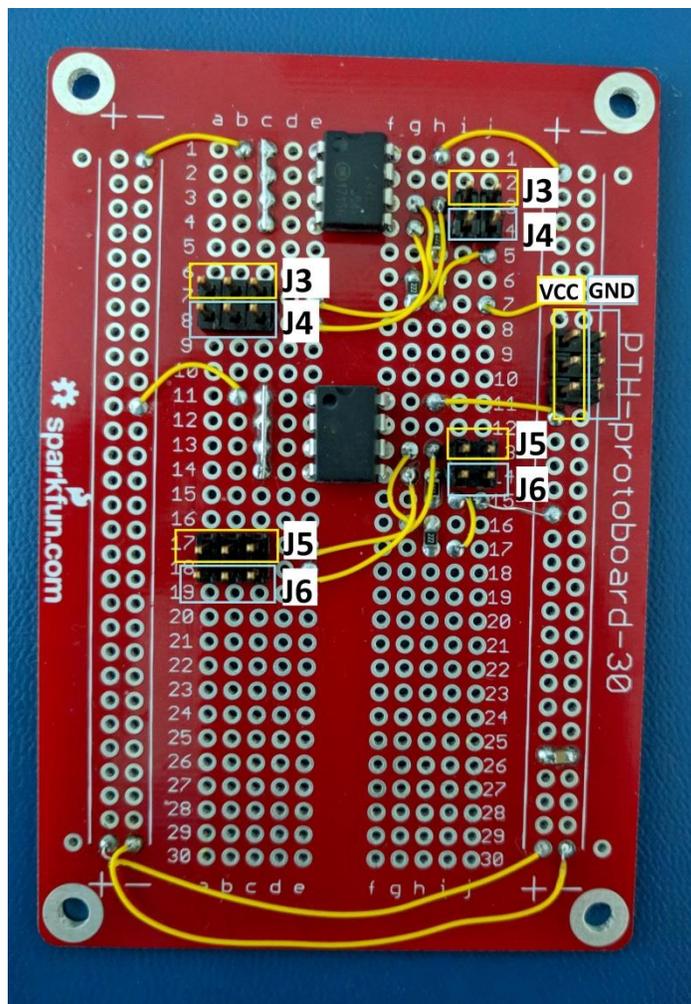


Figure 17. I2C EEPROM Test Board

The following table indicates the list of IO pins available on the board with descriptions.

Pins	Description
VCC	Vcc for board
GND	Gnd for board
J3	SCL Line for I2C0
J4	SDA Line for I2C0
J5	SCL Line for I2C1
J6	SDA Line for I2C1

Table 19. Test board pin details

5.2.2 I2C and SMBus IO Expander Evaluation Module

This module is available on [TI.com](http://www.ti.com). It is SMBus supporting IO Expansion module. The following is the picture of module. Further, information about the module is available on <http://www.ti.com/tool/io-expander-evm>.

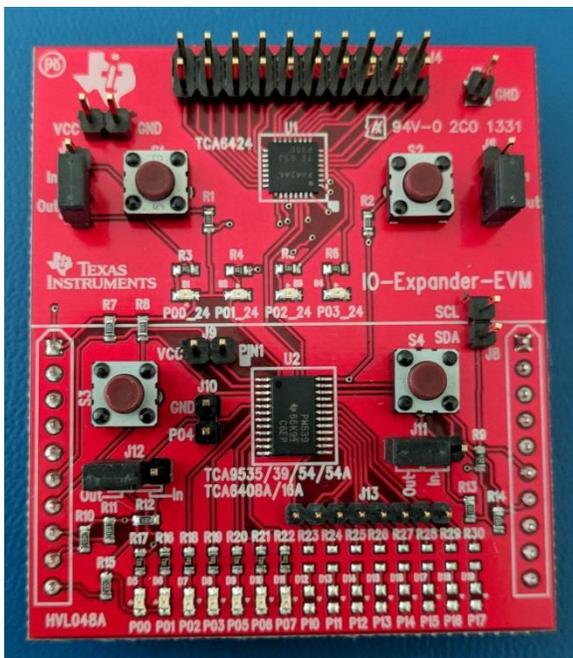


Figure 18. SMBus Expander module

5.3 Test Setup

5.3.1 Icev2AM335x

The following is the test setup connection required for the Firmware unit test. For this setup, you need to 1 EEPROM board and 1 SMBus expander module.

Icev2AM335x Pin	EEPROM Board Pin	SMBUS expander Pin
Port J3, Pin 1	VCC (any pin)	VCC
Port J3, Pin 2	GND (any pin)	GND
Port J3, Pin 14	J3 (any pin)	-
Port J4, Pin 21	J4 (any pin)	-
Port J3, Pin 12	J4 (any pin)	-
-	J3 (any pin)	SCL
-	J4 (any pin)	SDA

Table 20. icev2AM335x Test Setup

5.3.2 idkAM437x

The following is the test setup connection needed to be done for Firmware unit test. For this setup, you need to 1 EEPROM board and 2 SMBus expander module.

idkAM437x Pin	EEPROM Board Pin	SMBUS expander Pin
Port J16, Pin 1	VCC (any pin)	VCC (1 st Module)
Port J16, Pin 59	GND (any pin)	GND (1 st Module)
-	VCC (any pin)	VCC (2 nd Module)
-	GND (any pin)	GND (2 nd Module)
Port J3, Pin 6	J3 (any pin)	-
Port J3, Pin 5	J4 (any pin)	-
Port J3, Pin 8	J4 (any pin)	-
Port J16, Pin 46	J5 (any pin)	-
Port J3, Pin 7	J6 (any pin)	-
Port J16, Pin 48	J6 (any pin)	-
-	J3 (any pin)	SCL (1 st Module)
-	J4 (any pin)	SDA (1 st Module)
-	J5 (any pin)	SCL (2 nd Module)
-	J6 (any pin)	SDA (2 nd Module)

Table 21. idkAM437x Test Setup

5.3.3 idkAM572x

The following is the test setup connection required for the Firmware unit test. For this setup, you need to 1 EEPROM board and 1 SMBus expander module.

idkAM572x Pin	EEPROM Board Pin	SMBUS expander Pin
Port J21, Pin 1	VCC (any pin)	VCC
Port J21, Pin 60	GND (any pin)	GND
Port J21, Pin 5	J3 (any pin)	-
Port J46, Pin 4	J4 (any pin)	-
Port J21, Pin 3	J4 (any pin)	-
-	J3 (any pin)	SCL
-	J4 (any pin)	SDA

Table 22. idkAM572x Test Setup

5.3.4 idkAM574x

The following is the test setup connection required for the Firmware unit test. For this setup, you need to 1 EEPROM board and 1 SMBus expander module.

idkAM574x Pin	EEPROM Board Pin	SMBUS expander Pin
Port J21, Pin 1	VCC (any pin)	VCC
Port J21, Pin 60	GND (any pin)	GND
Port J21, Pin 5	J3 (any pin)	-
Port J46, Pin 4	J4 (any pin)	-
Port J21, Pin 3	J4 (any pin)	-
-	J3 (any pin)	SCL
-	J4 (any pin)	SDA

Table 23. idkAM574x Test Setup

5.4 Unit Test

The Unit Test checks all features of the I2C firmware. It tests all available instances, and all supported speeds. It prints a UART log during the execution of each test, an example of which is shown below.

I2C Test1: Instance 5: Baud Rate 100KHz:

Normal Read/Write test passed

SMBUS test passed

I2C Test2: Instance 5: Baud Rate 400KHz:

Normal Read/Write test passed

SMBUS test passed

I2C Test3: Instance 5: Baud Rate 1MHz:

Normal Read/Write test passed

All tests have passed.

6 Firmware Feature Enhancement

The I2C firmware described in previous sections of this document (I2C_FW) executes on AM437x ICSS1, but not AM437x ICSS0. This is because of the following hardware limitations of ICSS0 relative to ICSS1:

- PRU IMEM size is reduced from 8 to 4 kB. The I2C_FW program size is 5.74 kB.
- PRU DMEM size is reduced from 8 to 4 kB. Although the I2C_FW data memory size is only 3.25 kB = 0xD00 bytes, the starting offset of this data memory in DMEM is at location 0x400 (see Table 2). Hence the I2C_FW data memory spans 0x400 - 0x1100.
- There are no external connections to IEP pins (e.g. pr0_edio_data_out). As mentioned in Section 3.2.3, I2C_FW uses IEP DIGIO Output for SDA output.
- There is no Scratch Pad Memory (SPAD). I2C_FW uses SPAD for storing: (1) I2C instance context; and (2) the PRU0/1 local copy of the IEP DIGIO Output Enable register.

I2C_FW has been modified to execute from AM437x ICSS0 to provide additional flexibility in targeting I2C firmware to the available ICSS hardware resources on AM437x. This modified firmware (I2C_FW_AM437X_ICSS0) is described in this section.

6.1 Modifications to I2C Firmware for AM437X ICSS0

The modifications made to I2C_FW for I2C_FW_AM437X_ICSS0 include:

- SMBus support was removed. This reduced the firmware program memory size from 5.74 kB to 3.55 kB so the program fits within the 4 kB ICSS0 PRU IMEM.
- The firmware was updated to use the remote ICSS1 IEP DIGIO Output pins instead of the local ICSS0 IEP DIGIO pins since the ICSS1 IEP pins are available externally.
- DMEM was used in place of SPAD for storing I2C instance context. DMEM0 was used for instances executing on PRU0, while DMEM1 was used for instances executing on PRU1.
- DMEM was used in place of SPAD for storing the ICSS0 PRU0/1 local copies of the ICSS1 IEP DIGIO Output Enable register. DMEM0 was used for storing these local copies. However, the choice of DMEM0 is arbitrary, and the copies can alternately be stored in DMEM1 without incurring additional PRU cycles.
- The DMEM base address of the I2C configuration memory (see Table 2) was moved from 0x400 to 0x0 so the firmware data memory fits within the 4 kB ICSS0 PRU DMEM.

6.2 Features and Limitations of AM437X ICSS0 I2C Firmware

I2C_FW_AM437X_ICSS0 supports the same features as I2C_FW (see Table 1), with these exceptions:

- SMBus support is removed.
- HS mode (SCL clock frequency 1MHz) is currently unsupported.

It is not possible to simultaneously execute I2C_FW on ICSS1 and I2C_FW_AM437X_ICSS0 on ICSS0. This is because ICSS1 IEP DIGIO Output is used for SDA Output on both I2C_FW and I2C_FW_AM437X_ICSS0.

6.3 I2C Firmware Resource Requirements

6.3.1 Memory Requirements

The memory requirements for I2C_FW and I2C_FW_AM437X_ICSS0 on AM437x are presented in the table below.

Firmware	ICSS	PRU	IMEM (bytes used / available)	DMEM0 (bytes used / available)	DMEM1 (bytes used / available)	ICSS1 Shared Mem (bytes used / available)
I2C_FW	ICSS1	PRU0	0x16F8 /0x2000	0xD00 /0x2000	0 /0x2000	0
		PRU1	0x16F8 /0x2000	0 /0x2000	0xD00 /0x2000	0
I2C_FW_AM437X_ICSS0	ICSS0	PRU0	0xE30 /0x1000	0xDA8 /0x1000	0 /0x1000	0
		PRU1	0xE30 /0x1000	0 /0x1000	0xDA0 /0x1000	0

Table 24. I2C Firmware Memory Requirements

6.3.2 PRU Cycle Count Requirements

PRU cycle count data for I2C_FW and I2C_FW_AM437X_ICSS0 on AM437x for I2C Standard/Full modes is presented below. As discussed in Section 3.4.5, the I2C firmware

emulates I2C by dividing the I2C clock time into smaller time intervals (“time slice”), and performing I2C operations (e.g. driving SCL to a particular logic level) within this “oversampled” I2C clock time interval. The time slice interval is subdivided to provide support for multiple I2C instances. The PRU cycle counts for all firmware states must fit within the cycles for a subdivided time slice interval, i.e. all firmware state cycle counts must fit within the instance time slice PRU cycle budget. Hence the cycle count data below focuses on the maximum cycle count (C_{max}) across all firmware states processed for different types of I2C transactions. The cycle counts below were collected using the test program supplied with the PRSDK-RTOS I2C LLD.

6.3.2.1 I2C_FW, AM437X ICSS1

Standard Mode

Bus speed: 100 kHz

Oversampling (OS) factor per SCL clock cycle: 4

Number of supported I2C instances: 4

PRU cycles per I2C clock cycle: $200 \text{ MHz} / 100 \text{ kHz} = 2000$ cycles.

PRU cycles per I2C clock cycle per OS Time Slice (T_s): $2000 / 4 = 500$ cycles.

PRU cycles per I2C clock cycle per OS Time Slice per I2C instance (T_i): $500 / 4 = 125$ cycles.

Test Case	C_{max} State	C_{max}	C_{max}/T_s	C_{max}/T_i
eeeprom_write	ADDRESS_SDA_BEGIN	47	0.094	0.376
eeeprom_read	ADDRESS_SDA_BEGIN	47	0.094	0.376
loopback	RAISE_HOST_INTERRUPT_MEM_FOR_READY	46	0.092	0.368
test_probe	ADDRESS_SDA_BEGIN	47	0.094	0.376
test_probe_inv_addr	ADDRESS_SDA_BEGIN	47	0.094	0.376
eeeprom_write_buffer_ovr	RAISE_HOST_INTERRUPT_MEM_FOR_READY	46	0.092	0.368
bus_recovery_and_eeeprom_read	ADDRESS_SDA_BEGIN	47	0.094	0.376
timeout	ADDRESS_SDA_BEGIN	47	0.094	0.376

Table 25. I2C_FW Standard Mode Max. Cycle Counts (C_{max}) on AM437x ICSS1

Full Mode

Bus speed: 400 kHz

Oversampling (OS) factor per SCL clock cycle: 4

Number of supported I2C instances: 1

PRU cycles per I2C clock cycle: $200 \text{ MHz} / 400 \text{ kHz} = 500$ cycles.

PRU cycles per I2C clock cycle per OS Time Slice (T_s): $500 / 4 = 125$ cycles.

PRU cycles per I2C clock cycle per OS Time Slice per I2C instance (T_i): $125 / 1 = 125$ cycles.

Test Case	C_{max} State	C_{max}	C_{max}/T_s	C_{max}/T_i
eeeprom_write	ADDRESS_SDA_BEGIN	47	0.376	0.376
eeeprom_read	ADDRESS_SDA_BEGIN	47	0.376	0.376
loopback	RAISE_HOST_INTERRUPT_MEM_FOR_READY	46	0.368	0.368
test_probe	ADDRESS_SDA_BEGIN	47	0.376	0.376
test_probe_inv_addr	ADDRESS_SDA_BEGIN	47	0.376	0.376
eeeprom_write_buffer_ovr	RAISE_HOST_INTERRUPT_MEM_FOR_READY	46	0.368	0.368
bus_recovery_and_eeeprom_read	ADDRESS_SDA_BEGIN	47	0.376	0.376
timeout	ADDRESS_SDA_BEGIN	47	0.376	0.376

Table 26. I2C_FW Full Mode Max. Cycle Counts (C_{max}) on AM437x ICSS1

HS Mode

Bus speed: 1 MHz

Oversampling (OS) factor per SCL clock cycle: 4

Number of supported I2C instances: 1

PRU cycles per I2C clock cycle: 200 MHz / 1 MHz = 200 cycles.

PRU cycles per I2C clock cycle per OS Time Slice (T_s): 200/4 = 50 cycles.

PRU cycles per I2C clock cycle per OS Time Slice per I2C instance (T_i): 50/1 = 50 cycles.

Test Case	C_{max} State	C_{max}	C_{max}/T_s	C_{max}/T_i
eeeprom_write	ADDRESS_SDA_BEGIN	34	0.68	0.68
eeeprom_read	ADDRESS_SDA_BEGIN	34	0.68	0.68

Table 27. I2C_FW HS Mode Max. Cycle Counts (C_{max}) on AM437x ICSS1.

Note: cycle counts were only collected for EEPROM write and read transactions for HS mode since the pattern of cycle counts for the other transaction types is expected to be consistent with those collected for Standard and Full modes. In particular, C_{max} is not expected to change for the other transaction types.

6.3.2.2 I2C_FW_AM437X_ICSS0, AM437X ICSS0

Standard Mode

Test Case	C_{max} State	C_{max}	C_{max}/T_s	C_{max}/T_i
eeeprom_write	ADDRESS_SDA_BEGIN	77	0.154	0.616
eeeprom_read	ADDRESS_SDA_BEGIN	77	0.154	0.616
loopback	RAISE_HOST_INTERRUPT_MEM_FOR_READY	74	0.148	0.592
test_probe	ADDRESS_SDA_BEGIN	77	0.154	0.616
test_probe_inv_addr	ADDRESS_SDA_BEGIN	77	0.154	0.616
eeeprom_write_buffer_ovr	RAISE_HOST_INTERRUPT_MEM_FOR_READY	74	0.148	0.592
bus_recovery_and_eeeprom_read	ADDRESS_SDA_BEGIN	77	0.154	0.616
timeout	ADDRESS_SDA_BEGIN	77	0.154	0.616

Table 28. I2C_FW_AM437X_ICSS0 Standard Mode Max. Cycle Counts (C_{max}) on AM437x ICSS0

Full Mode

Test Case	C_{max} State	C_{max}	C_{max}/T_s	C_{max}/T_i
eeeprom_write	ADDRESS_SDA_BEGIN	77	0.616	0.616
eeeprom_read	ADDRESS_SDA_BEGIN	77	0.616	0.616
loopback	RAISE_HOST_INTERRUPT_MEM_FOR_READY	74	0.592	0.592
test_probe	ADDRESS_SDA_BEGIN	77	0.616	0.616
test_probe_inv_addr	ADDRESS_SDA_BEGIN	77	0.616	0.616
eeeprom_write_buffer_ovr	RAISE_HOST_INTERRUPT_MEM_FOR_READY	74	0.592	0.592
bus_recovery_and_eeeprom_read	ADDRESS_SDA_BEGIN	77	0.616	0.616
timeout	ADDRESS_SDA_BEGIN	77	0.616	0.616

Table 29. I2C_FW_AM437X_ICSS0 Full Mode Max. Cycle Counts (C_{max}) on AM437x ICSS0

HS Mode

HS mode (SCL clock frequency 1MHz) is currently unsupported on ICSS0.

6.3.2.2.1 I2C_FW_AM437X_ICSS0 Maximum Cycle Counts Details

Standard/Full Modes

I2C_FW max(C_{max}): 47

I2C_FW_AM437X_ICSS0 max(C_{max}): 77

$$77 = 47 + (11+10) + (3+2) + 4$$

- 47: I2C_FW max(C_{max}), max. before any firmware updates
- (11+10): cycles added for context restore/save using DMEM instead of SPAD
- (3+2): cycles added for using DMEM instead of SPAD for PRU0/1 copies of IEP DIGIO Output Enable.
- 4: cycles added for remote access to ICSS1 IEP from ICSS0

Total added cycles for ICSS0: $77 - 47 = 30$.

Total cycles added for SPAD replacement w/ DMEM: $26. 26/30*100 = 86.7\%$.

Table 28 shows I2C_FW_AM437X_ICSS0 max(C_{max}) fits within the instance time slice PRU cycle budget for Standard mode @ 100 kHz. Similarly, Table 29 shows I2C_FW_AM437X_ICSS0 max(C_{max}) fits within the instance time slice PRU cycle budget for Full mode @ 400 kHz.

HS Mode

HS mode (SCL clock frequency 1MHz) is currently unsupported on ICSS0.

6.4 Test Plan

6.4.1 EVM Support

6.4.1.1 idkAM437x

The following table provides details concerning the AM437x IDK expansion header pins assigned to I2C instances for I2C_FW_AM437X_ICSS0 and I2C_FW. This table was derived

from Table 16. The last six rows were added for the IDK expansion header pins assigned to I2C_FW_AM437X_ICSS0 I2C instances.

ICSS	PRU	Instance	Functional Pin	PRU GPIO Pins	EVM Port	EVM pin
ICSS1	PRU0	I2C0	SCL	pr1_pru0_pru_r30_8	J3	6
			SDA	pr1_edio_data_out0	J3	5
				pr1_pru0_pru_r31_9	J3	8
		I2C1	SCL	pr1_pru0_pru_r30_10	J16	46
			SDA	pr1_edio_data_out1	J3	7
				pr1_pru0_pru_r31_11	J16	48
ICSS0	PRU0	I2C0	SCL	pr0_pru0_pru_r30_8	J16	56
			SDA	pr1_edio_data_out0	J3	5
				pr0_pru0_pru_r31_9	J16	37
		I2C1	SCL	pr0_pru0_pru_r30_10	J16	38
			SDA	pr1_edio_data_out1	J3	7
				pr0_pru0_pru_r31_11	J16	58

Table 30. idkAM437x I2C Instances

6.4.2 Test Setup

6.4.2.1 idkAM437x

The test setup connections required for the I2C firmware unit test are presented in the following table. This setup is used for testing the I2C_FW_AM437X_ICSS0 binaries for ICSS0, as well as the I2C_FW binaries for ICSS1. The setup uses the same external I2C boards covered in Section 5.2. For this setup, 1 EEPROM board and 2 SMBus expander modules are needed. The EEPROM board and SMBus expander modules are used for testing I2C_FW binaries, while only the EEPROM board is used for testing of the I2C_FW_AM437X_ICSS0 binaries.

The table below was derived from Table 21. The last four rows were added for the connections required for I2C_FW_AM437X_ICSS0 testing, and the last column was added to describe which binaries require the connection in each row.

idkAM437x Pin	EEPROM Board Pin	SMBus expander Pin	Used by ICSS1/0 binaries
Port J16, Pin 1	VCC (any pin)	VCC (1 st Module)	ICCS1 & ICSS0
Port J16, Pin 59	GND (any pin)	GND (1 st Module)	ICCS0 & ICSS0
-	VCC (any pin)	VCC (2 nd Module)	ICSS1
-	GND (any pin)	GND (2 nd Module)	ICSS1
Port J3, Pin 6	J3 (any pin)	-	ICSS1
Port J3, Pin 5	J4 (any pin)	-	ICSS1 & ICSS0
Port J3, Pin 8	J4 (any pin)	-	ICSS1
Port J16, Pin 46	J5 (any pin)	-	ICSS1
Port J3, Pin 7	J6 (any pin)	-	ICSS1 & ICSS0
Port J16, Pin 48	J6 (any pin)	-	ICSS1
-	J3 (any pin)	SCL (1 st Module)	ICSS1
-	J4 (any pin)	SDA (1 st Module)	ICSS1
-	J5 (any pin)	SCL (2 nd Module)	ICSS1
-	J6 (any pin)	SDA (2 nd Module)	ICSS1
Port J16, Pin 56	J3 (any pin)	-	ICSS0
Port J16, Pin 37	J4 (any pin)	-	ICSS0
Port J16, Pin 38	J5 (any pin)	-	ICSS0
Port J16, Pin 58	J6 (any pin)	-	ICSS0

Table 31. idkAM437x Test Setup

6.4.3 Unit Test

The I2C_FW_AM437X_ICSS0 Unit Test checks all features of the I2C firmware. It tests all available instances, and all bus speeds. It prints a UART log during the execution of each test, an example of which is shown below.

I2C Test1: Instance 3: Baud Rate 100KHz:

Normal Read/Write test passed

I2C Test2: Instance 3: Baud Rate 400KHz:

Normal Read/Write test passed

I2C Test3: Instance 3: AM437x ICSS0: Baud Rate 781.25 kHz:

Normal Read/Write test passed

I2C Test4: Instance 4: Baud Rate 100KHz:

Normal Read/Write test passed

All tests have passed.

I2C_FW Unit Test is unchanged, and is described in Section 5.4.